SwisTrack - A Flexible Open Source Tracking Software for Multi-Agent Systems

Thomas Lochmatter, Pierre Roduit, Chris Cianci, Nikolaus Correll, Jacques Jacot and Alcherio Martinoli

***Abstract*— Vision-based tracking is used in nearly all robotic laboratories for monitoring and extracting of agent positions, orientations, and trajectories. However, there is currently no accepted standard software solution available, so many research groups resort to developing and using their own custom software. In this paper, we present Version 4 of *SwisTrack*, an open source project for simultaneous tracking of multiple agents. While its broad range of pre-implemented algorithmic components allows it to be used in a variety of experimental applications, its novelty stands in its highly modular architecture. Advanced users can therefore also implement additional customized modules which extend the functionality of the existing components within the provided interface. This paper introduces *SwisTrack* and shows experiments with both marked and marker-less agents.**

I. Introduction

Tracking moving agents with overhead cameras is an technique often used in robotics and other related research areas. Cameras are relatively cheap and easy to use, and therefore provide easily accessible "ground truth" measurements and indoor localization. Such data can be very valuable for offline analysis, e. g. to analyze trajectories [1], to measure traveled distances, or simply to check whether an experiment was successful or not. In addition, such systems can give robots online feedback about their current position, emulating an indoor GPS, for instance.

Even though cameras are frequently used in nearly all robotic labs, it seems that no single common integral software package exists. Vicon Motion Systems provides a solution [2] efficient enough for most robotics application, but it is hardly affordable for most research laboratories. Noldus Information Technologies offers another commercial option with EthoVision [3], but neither solution can be tailored to non-standard custom applications or environments, as their source code bases cannot be modified by the end user.

As a result, many robotic research labs therefore write their own tracking software more or less from scratch. At first glance, the task seems to be fairly simple, and with a few lines of code one can arrive at a crude, but serviceable, solution; which would seem to imply that a proper implementation should not take more than a few dozens of additional hours of work. However, when such minimal effort is invested, the resulting program is rarely usable by anyone other than the original programmer or in any context other than the original experiment. Typically, if a graphical user interface is implemented, it is extremely rudimentary, with parameters being specified directly in the source code (necessitating recompilation for even simple adjustments). In an active research laboratory, with researchers joining and leaving on a regular basis, this process is repeated over and over, summing up to an incredible amount of man-hours for what was previously thought to be a "simple" tool. RoboTracker [4] is an example of a tracking application built from scratch and not accessible to the scientific community. Other research laboratories are diffusing their work, such as TrackIt [5] or XVision [6]. Open source software does exist, but was developed for specific applications, such as activity monitoring [7], [8] or tracking of specific markers [9]. Many of these solutions are bound to a specific camera interface, and the option to do off-line tracking using stored video is uncommon. In addition, none of them provide a modular architecture. ImageJ [10], an image processing utility with a plugin-based interface, is an example of this type of shared resource which has had a profound impact on its community.

Correll et al. [11] therefore started an open source project called *SwisTrack* to serve as a generic and flexible tool for tracking robots and insects. Version 3 of this program, released in 2006, has been adopted by several other research laboratories, and the community behind it continues to grow; around 60 to 100 downloads occur each month from the Sourceforge website. Some examples of its use include the tracking of Khepera III and e-puck mobile robots in flocking and formation experiments by the *Departamento de Tecnologías Especiales Aplicadas a la Telecomunicación* and the *División de Ingeniería de Sistemas y Automática* of the *Universidad Politécnica de Madrid*, and the tracking of young chickens in the *Miniature Mobile Robots Group* from the *École Polytechnique Fédérale de Lausanne*.

In this paper, we present version 4 of *SwisTrack*. It has been completely rewritten and expanded, using the knowledge and experience acquired from working with the previous versions. Noteworthy, in particular, is the new modular architecture and the considerable amount of implemented algorithmic components provided. The resulting program is a stable, production quality tracking package for both marked and markerless agents, with support for several different

T. Lochmatter, P. Roduit, C. Cianci and A. Martinoli are with the Distributed Intelligent Systems and Algorithms Laboratory, P. Roduit and J. Jacot are with the "Laboratoire de Production Microtechnique", both at the École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland. N. Correll is with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 02139 Cambridge, MA, USA. (thomas.lochmatter@epfl.ch, pierre.roduit@epfl.ch)

This work was partly supported by the National Competence Center in Research on Mobile Information and Communication Systems NCCR-MICS, a center supported by the Swiss NSF under grant number 5005-67322. P. Roduit and C. Cianci are currently sponsored by Swiss NSF grant contracts Nr. 200020- 113795 and PP002–68647 respectively.

Component	Trigger	Input	Color image	Grayscale image	Binary image	Particles	Tracks	Messages
Threshold a grayscale image				R	W			
Binary mask					E			

Fig. 1. A close-up view of the processing pipeline of *SwisTrack*'s main window (see Figure 2). Data channels are displayed as columns and can be read (R), edited (E) and written (W) by components. In this example a grayscale image is thresholded, which yields a binary image. The binary image is then modified by the *Binary mask* component.

camera standards. It can also be used on both Windows and Linux systems. A graphical user interface allows the user to set up a processing pipeline consisting of components (modules implementing processing steps). The parameters of each component can then be changed at runtime, and component output can be visualized in real-time on the screen. *SwisTrack* also keeps track of the processing time taken by each operation, which can be very helpful for speed optimizations.

SwisTrack can also be used to record agent trajectories during an experiment, either from a camera (in real time) or a segment of recorded video (post-processing). For example, *SwisTrack* can be used to feed position information to the robots (or other devices) during the experiment, and data can be saved or output in a variety of formats for later use.

In addition, *SwisTrack* can easily be extended by implementing new modular components, which can subsequently be used and configured just as any other component. Therefore, even very specific tracking applications can make use of the existing *SwisTrack* framework.

SwisTrack's documentation is written as a Wikibook [12] and contains—in addition to general usage hints—a series of examples which may serve as a good starting point for any of a number of possible tracking setups.

The remainder of this paper is organized as follows: Section II gives an overview of *SwisTrack* and its components. Section III explains how *SwisTrack* can be used in a multi-camera setup with coded markers. This is followed by two single-camera experiments (Section IV), and a multi-camera experiment (Section V).

II. Software Overview

In this section, we give a brief overview of *SwisTrack*'s architecture. This architecture has been remodeled since the initial release [11] in order to allow easy combination of new and existing algorithms, input, and output media.

The software architecture is component-based, with each component conforming to well-defined interfaces for interaction with the other components in the pipeline. Components pass their data through structures referred to here as "data channels."

The existing data channels, shown in Figure 1, are: *input*, *grayscale image*, *color image*, *binary image*, *particles* and *tracks*. A component can read (**R**), edit (**E**) or write (**W**) any of these data channels, as illustrated in Figure 1. For example, the component *Threshold a grayscale image* will read the *grayscale image* channel, threshold the image (i. e. give a value of 1 to all pixels with an intensity within

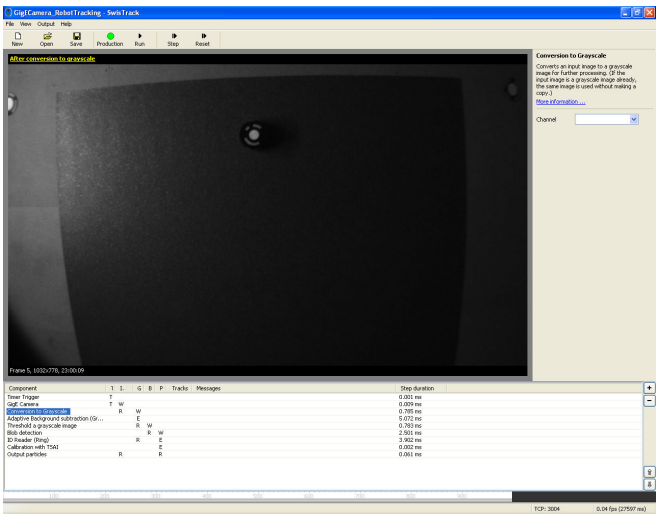


Fig. 2. *SwisTrack*'s main window with the display in the center, the processing pipeline and the timeline at the bottom, and the configuration panel on the right side. The configuration panel displays the controls and parameters associated with each component.

given boundaries and 0 to all other pixels), and write the resulting binary image into the *binary image* channel. Then, the *Binary mask* component reads from the *binary image* channel, applies a logical AND ($\wedge$) operation with the desired mask image, and writes the result back into the *binary image* channel. This component is therefore marked as editing (**E**) the *binary image* channel.

A *SwisTrack* project mainly consists of an ordered set of components (the processing pipeline) assembled by the user. Figure 3 shows an example of such a pipeline. Each component in this pipeline has a series of parameters which can be modified in *SwisTrack*'s right-hand panel. When processing a camera frame, the components are applied (in order) to the acquired frame with their current configuration. After each component, the resulting processed image can be displayed on the screen. There is no limit in the number of components and their order. The user needs only to ensure that the data channel read by any given component has previously been written by another component. The same component can also be applied multiple times, if desired (even if this does not make sense for most components).

For catering to most typical tracking scenarios, *SwisTrack* classifies the components into ten categories, and proposes a certain order in which these components should be executed. The categories along with a description of the most important algorithms therein are listed in their proposed execution order

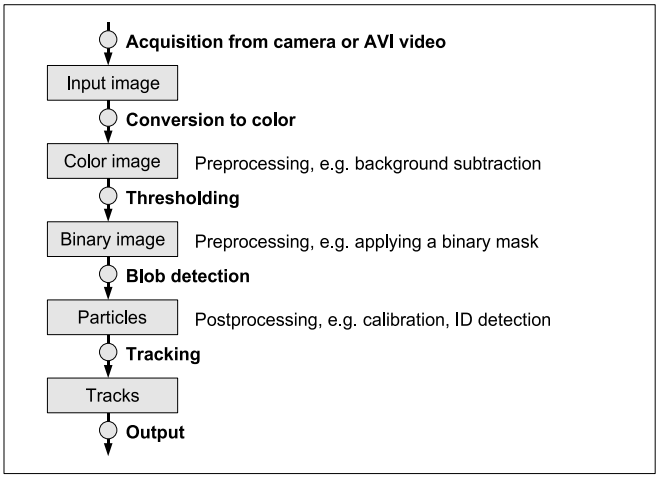


Fig. 3. A standard processing pipeline for color images. The pipeline is similar for grayscale images.

in Table I. This classification supposes that the input image is transformed into a binary image (after some preprocessing steps), and that the objects to be tracked are detected as blobs (i. e. set of connected pixels with the same value in a binary image). Blobs are then converted into particles (i. e. points). The particles of successive frames can be associated to tracks, and sent to an output component. This general structure has emerged from experiences with different requirements in our group and elsewhere.

A. Visual Feedback for Parameter Tuning

Most of the components modify an image data channel and thus allow the user to visualize its results online. Relevant parameters of each component can be modified on the fly, which allows for fast prototyping of entire tracking applications and easy adaption to changing environment conditions.

B. Interfacing SwisTrack

SwisTrack provides two ways to output data: an *output to file* component which writes tab-separated files, and a TCP interface with NMEA 0183 message streaming. NMEA 0183 is a simple human-readable and easily parsable protocol, well known from GPS receivers. Code examples for C, C++, Java, Perl and Python are provided. The TCP interface not only allows several clients to receive information about the object's positions, but also to remote control *SwisTrack*.

C. Implementing Specific Components

While the already available components allow for catering to a large variety of tracking scenarios, it may sometimes be necessary to implement a specific algorithm, e. g. to track a special type of object, deal with a special type of environment, or simply acquire images from a new type of camera (or camera driver).

This can be done by implementing a new component in *SwisTrack*. For easing this process, a component template is available, and detailed information is available in the wikibook [12]. Relevant parameters and their data types of each component are stored in an XML file, which allows *SwisTrack* to automatically generate a graphical user interface specific to the custom component.

Note that even for very special types of object tracking, we believe that implementing a component in *SwisTrack* should be preferred over writing a separate program, as *SwisTrack* allows to leverage on a suite of useful features (camera support, displaying images, calibration, ...) and simplifies the debugging process with its graphical user interface and its strong encapsulation.

III. Multi-Camera Tracking and Coded Markers

SwisTrack has recently been endowed with a multi-camera tracking feature. Using more than one camera adds additional challenges. First, robots can disappear on a camera image an reappear later, and second, robots shall be tracked when they leave one camera range and enter another.

Merging all camera images and doing the detection on one big image is clearly very expensive in terms of processing and memory requirements. A much more viable approach is to detect the robots on all camera images individually, and to merge the robot positions afterward. This requires some camera range overlap to make sure that the trajectories are merged correctly when a robot moves from one camera image to another.

A. Coded Markers

When working with multiple robots, an additional increase in robustness can be achieved by marking the robots with a unique code. This allows the tracker to maintain a continuous trajectory even if the robot disappears temporarily (e. g., because of occlusion). We developed a set of markers using a 14-chip circular code. All codes are rotationally unique, i. e. no two codes are the same in any rotation relative to each other. To make the detection more robust, the codes have a Hamming distance of at least 4 chips in any rotation. Using a simple heuristic algorithm, we found 25 such 14-chip codes. A fair number of different marking strategies have been developed by the scientific community, such as ARTag [13], but in our case the circular patterns help us to get a precise positioning.

To use such codes, the camera needs to have a rather high resolution, i. e. the code should occupy an area of at least 20 by 20 pixels on a good quality image.

To decode the ID, the pixels on the ring are first transformed into a vector of brightness and angle values. The covariance between this vector and all expected codes (in different rotations) then reveals which code is seen, under which angle it is seen (with 2 - 10 degrees accuracy), and a confidence level (covariance value). The latter can be used to filter out blobs that were by mistake detected by the blob detection algorithm.

IV. Single-Camera Experiments

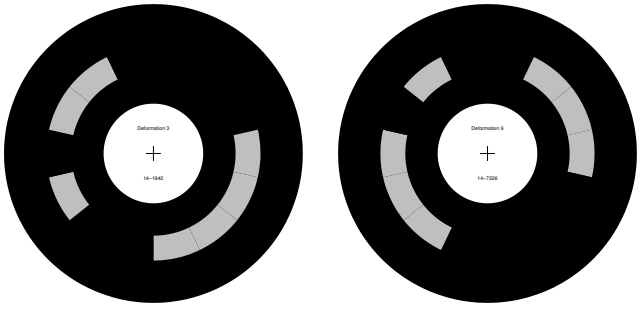
In a first series of experiments, we demonstrate how the available components of *SwisTrack* can be wired together for tracking agents with and without markers using a single camera.

TABLE I

COMPONENT CATEGORIES.

	Category	Reads	Writes	Description of the main algorithms implemented in the current components
1	*Trigger*	-	-	Tell *SwisTrack* when to process the next frame, e. g. using a timer
2	*Input*	-	I	Image acquisition from a camera (USB, FireWire, Basler GigE) or a file (AVI, BMP, ...)
3	*Input conversion*	I	G, C	Conversion to color or grayscale, or from a Bayer pattern
4a	*Preprocessing (color)*	C	C	Color image transformations, e. g. background subtraction, fixed color subtraction, channel arithmetic
4b	*Preprocessing (grayscale)*	G	G	Grayscale image transformations, e. g. background subtraction
5	*Thresholding*	G, C	B	Conversion to a binary image using a multi-channel or single-channel threshold
6	*Preprocessing (binary)*	B	B	Binary image operations, such as erosion, dilation, masking, and blob filtering
7	*Particle detection*	B	P	Blob detection on the binary image (marker-less, marker-based with or without codes)
8	*Calibration*	P	P	Transformation from pixel coordinates to world coordinates, using a second-order linear model or Tsai's algorithm
9	*Tracking*	P	T	Association of particles with trajectories, using a nearest neighbor criterion
10	*Output*	All	-	Output to file (tab-separated) or TCP connection (NMEA 0183 messages)

Data channels: **I** = Input, **G** = Grayscale image, **C** = Color image, **B** = Binary image, **P** = Particles, **T** = Tracks



ID 3 ID 9

Fig. 4. Two sample markers. The white blob in the middle allows for the detection of the robot's position. The circular code around allows to discriminate between up to 25 robots, and to detect their orientation.

A. Marker-based tracking

As first example we consider tracking of 8 e-puck robots [14] that are endowed with elliptic markers in a setup with one overhead camera.

1) Setup: A white arena of 1.60 by 1.60 m with 8 e-puck robots is captured with one GigE color camera (Basler Vision Technologies) mounted on the ceiling. The arena, depicted in Figure 5, covers a size of roughly 660 by 660 pixels on the image.

Special attention was given to the light. The light in the room is diffuse, which ensures an even illumination of the arena and greatly helps reducing dark shadows.

Each robot is carrying a circular piece of white paper with a black ellipse drawn on it. Such an ellipse has a size of about 13 by 30 pixels on the image. Attaching the marker in the center of the robot helps to prevent that blobs can seemingly merge. Whether a simple nearest-neighbor algorithm is then sufficient for tracking is a function of the robot speed and the acquisition frame rate.

2) SwisTrack Configuration: SwisTrack is configured as follows. A frame is acquired with a GigE camera component, converted to grayscale and thresholded. Since the background is white and homogeneous, there is no need to apply background subtraction here. Then, blob detection is applied with two selection criteria: the blobs must be within a certain size range, and be reasonably compact. The effect of this selection is demonstrated in Figure 6. After thresholding, not only the blobs, but also robot shadows and the arena borders are visible on the image. While the arena border and other markings are filtered out with the size criterion, the shadows have a compactness below the selection criterion. The resulting image contains only the markers. Finally, the robots are tracked with the nearest neighbor tracking component.

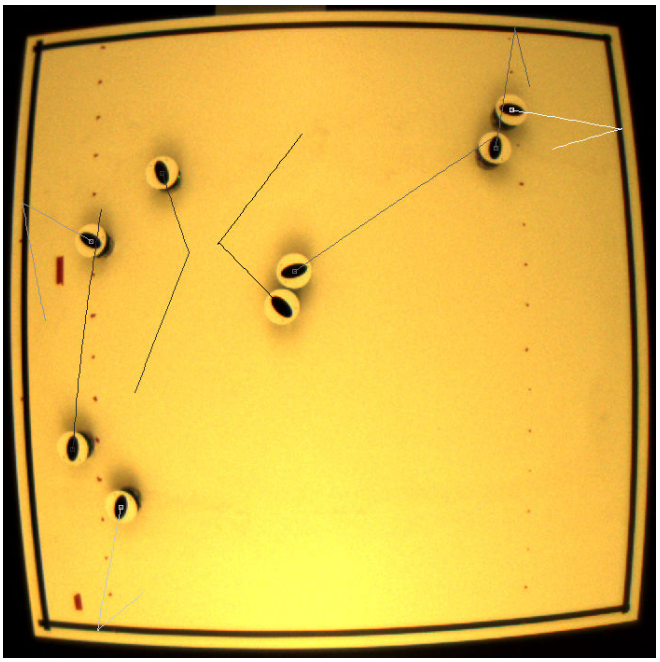
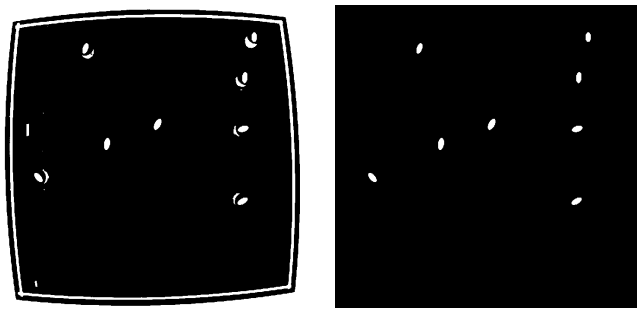


Fig. 5. An image of the overhead camera showing the 8 e-pucks in the arena. Robots are equipped with elliptical markers.

3) Results: Both blob detection and nearest neighbor tracking work reliably in the whole arena. The robot posi-



(a) (b)

Fig. 6. A sample image after thresholding (a) and after blob selection (b). Blobs are selected by their compactness and size, which is very robust in this setup.

tions are detected at a precision below 2 mm. In this setup, it would be possible to work without any markers on the robots, if one does not need the additional position accuracy that they provide. Since the blobs are ellipses, the blob detection algorithm also reliably detects their orientation with an error of maximum 5 degrees.

B. Marker-less tracking

SwisTrack has been also used for tracking cockroaches [15]. Tracking cockroaches is particularly challenging as the animals can move extremely fast (up to 50 cm/s), tend to stay very close together or on top of each other, and have widely varying sizes and appearances when compared with robotic agents (see also [11]).

1) Setup: Cockroaches walk in a circular arena of about 1 meter of diameter, shown in Figure 7(a). An overhead camera was used to produce AVI video files (MPEG-4) with a framerate of 30 Hz recording the experiment. Pictures of the empty arena and of a calibration pattern with 69 dots were saved at the same time.

2) SwisTrack Configuration: Images are acquired from an AVI video feed and the image is converted to a grayscale image. Then, the image of the empty arena (background) is subtracted from the input frames of the video, and a simple threshold is applied on the resulting gray image. The threshold value is estimated online by the user using the detected blobs as visual feedback. A binary mask consisting of the arena shape is then applied on the resulting binary image to remove all the objects outside of the arena. Next, a close operation (an erode followed by dilate) is applied to fill small holes inside of the blobs. The resulting blobs are filtered based on their size to remove blobs that are too small or too big to be a cockroach. The direct visual feedback of the filter results eases parameter tuning substantially here. The center of mass of the blobs then become the particle positions, which are subsequently transformed from pixel coordinates to metric space using Tsai's algorithm [16]. Finally, a nearest neighbor tracking method is used to create trajectories from the extracted particles, as shown in Figure 7(c).

3) Results: The error of a calibration method can be quantified as the distance between the real position and the calculated positions of the points used in the calibration process.

Due to the large optical deformation in this setup, Tsai's calibration method [16] yields particularly good results. The mean value of the error on the 69 calibration points was only 0.4 mm, and the maximal error was 1 mm. This corresponds to a relative error of about 0.5 $^0/_{00}$ for our arena diameter of 1 m.

For comparison, calibration with a second order linear model yields a mean error of 3.9 mm and a maximal error of 6.0 mm.

V. Multi-Camera Experiments

In a second series of experiments, we consider tracking Khepera III robots [17] endowed with coded markers in a multi-camera setup.

A. Setup

The arena is tracked by two overhead cameras, each connected to its own computer. Both computers run an instance of *SwisTrack*. A small script intercepts the output of both instances, and records it for offline processing. The arena monitored by two GigE color cameras has a size of approximately 3.4 m by 2.6 m. The cameras are synchronized by an external trigger signal and have a resolution of 1032 by 778 pixels, resulting in a pixel size of about 2.5 mm by 2.5 mm on the floor. For demonstration purposes, a large overlap region of about 80 cm width is chosen, as shown in Figure 9. The ceiling is low, requiring and a wide angle lens with 3.6 mm focal length, which in turn results in a visible distortion at the image borders. All robots are equipped with the 14-chip coded markers described in Section III-A, which are projected onto about 40 by 40 pixels on the image.

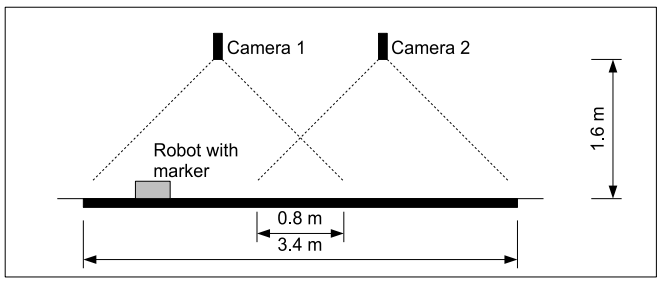
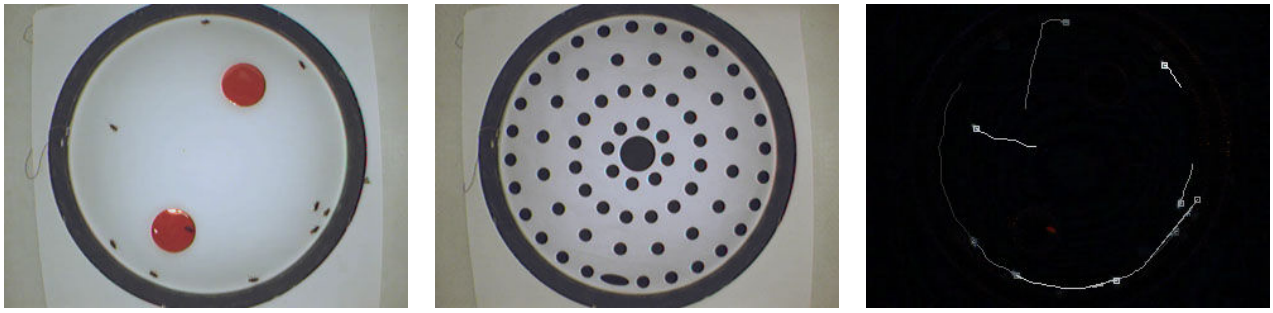


Fig. 9. A sketch (not to scale) of the setup with two cameras.

B. SwisTrack Configuration

SwisTrack is configured as follows. A frame is acquired with a GigE camera component and converted to grayscale (Figure 8 (a)). The image is then passed through an adaptive background subtraction component (Figure 8 (b)), initialized at the beginning with no robot in the arena, and then thresholded (Figure 8 (c)). On the black and white image, blob detection is applied (Figure 8 (d)). The blobs are filtered by their size and their compactness, which allows us to get rid of most of the occasional bright reflections on the robot. Finally, the ID is read (Figure 8 (e)) and the robot



(a) Top view of cockroaches moving in a closed arena.

(b) Pattern used for the calibration.

(c) Trajectories resulting of the cockroaches tracking.

Fig. 7. Tracking of cockroaches in a circular arena.

(a) (b) (c) (d) (e)

Fig. 8. A robot as seen from the camera (a), after background subtraction (b), after thresholding (c), after blob detection (d) and after reading its ID (e) in the multi-camera setup.

positions on the image are transformed to world coordinates. The calibration process is explained below.

The procedure is exactly the same for both cameras, but the parameters are sometimes slightly different to compensate for the minor differences in light conditions on both sides of the arena.

C. Calibration

The detected robot positions are calibrated using Tsai's calibration method [16], an algorithm requiring information on the position of at least 5 non-collinear points on the image. Such points were found by driving a robot to various locations in the arena, and by noting their position on the camera as well as their "real" position. This "real" position was measured using the odometry of the (differential-drive) robot, which is very precise over the short distances (few meters) involved in this calibration procedure.

Both cameras were calibrated in a single operation, i. e. whenever the robot reached one calibration point, an image was taken with both cameras. One camera saw the robot 8 times, while the other camera detected it 9 times in its range. Two calibration points were in the overlap region and therefore shared by both cameras.

Thanks to *SwisTrack*'s TCP interface, such procedures can be fully automatized with a few simple scripts.

D. Results

Figure 10 shows a robot trajectory in the overlap region. The distance between the trajectories recorded by the two cameras is in the order of 1-2 cm, which is precise enough for most purposes. Given that the robots have a diameter of roughly 12 cm, the two trajectories can easily be matched using a nearest neighbor criteria at each time step.

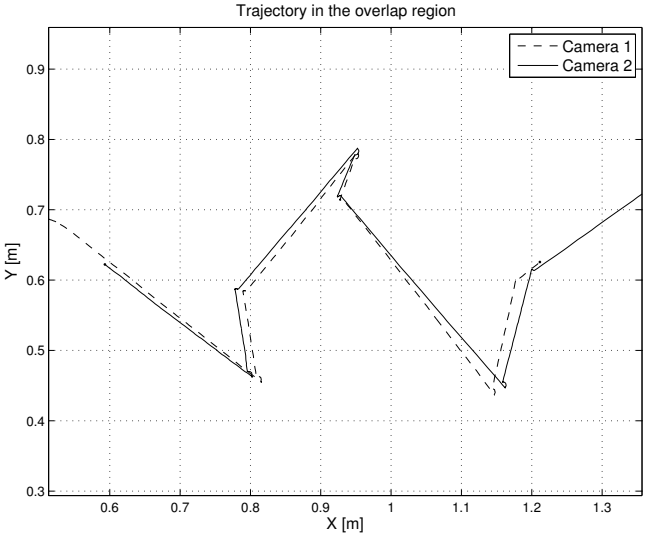


Fig. 10. A sample trajectory of the overlap region, generated by an odor source localization algorithm [17].

In the presented setup, the ID is always correctly decoded, except at the border of the image where the code is not completely visible. Furthermore, the detected robot orientation angle is within 2 degrees precision. Since the orientation is not corrected by the calibration component, its accuracy is worse, though.

The complete processing of one frame takes about 20 ms on a state-of-the-art computer. Since images were acquired at 20 fps, the processor is only moderately loaded. The most

time-consuming operations are blob detection and reading the code on the marker.

VI. Conclusion

We have introduced the new version of *SwisTrack*, a stable software solution for multi-agent tracking, and have shown its flexibility through three experiments with robots and cockroaches. Nonetheless, *SwisTrack* is not limited to the examples presented here and offers modularity, extendability, and interoperability making it an attractive tool for any kind of agent tracking.

Simple problems, such as tracking a robot with an overhead camera, can be solved with the existing components, without writing a single additional line of code. For more complex problems, the implementation of a new component may be necessary. But even then, *SwisTrack* provides an invaluable service, as it contains a complete graphical interface, a clear structure, and many re-usable components that solve parts of the problem. For example, agent tracking in a complicated environment may require implementing an advanced agent detection algorithm, but can fully reuse *SwisTrack*'s image acquisition, preprocessing, tracking, and output components.

The *SwisTrack* development team has benefited from the knowledge of both a computer vision and a robotics research group. While the implemented algorithms are not necessarily novel, their compilation into a single stable tracking program is.

We believe that *SwisTrack* greatly supports research efforts involving cameras. Therefore, we encourage research groups to take part in the collaborative effort by using the software, and potentially implementing and contributing their own algorithmic components. The user interface is straightforward; the average moderately skilled computer user is able to become productive within a very short period of time. Furthermore, with the help provided in the *SwisTrack* documentation [12], a good C++ programmer should be able to implement components with relative ease.

By publishing *SwisTrack* as an open source program, we hope to alleviate the current unfortunate state of affairs in which many research groups have felt obligated to re-invent their own solution due to the lack of a readily-available standard for multi-agent tracking.

References

[1] P. Roduit, A. Martinoli, and J. Jacot, "A Quantitative Method for Comparing Trajectories of Mobile Robots Using Point Distribution Models," in *Proc. of the 2007 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2007, pp. 2441–2448.

[2] Vicon Motion Systems. (2008, June) Vicon MX. [Online]. Available: http://www.vicon.com/products/viconmx.html

[3] Noldus Information Technology. (2008, June) Ethovision 3.1. [Online]. Available: http://www.noldus.com/site/doc200801005

[4] A. Sirota, "Robotracker - a system for tracking multiple robots in real time," Technion Israel Institute of Technology, Tech. Rep., December 2004.

[5] S. N. Fry, M. Bichsel, P. Mller, and D. Robert, "Tracking of flying insects using pan-tilt cameras," *Journal of Neuroscience Methods*, vol. 101, no. 1, pp. 59–67, August 2000.

[6] Computational Interaction and Robotics Lab - John Hopkins University. (2008, June) A Brief Tour of XVision. [Online]. Available: http://www.cs.jhu.edu/CIPS/xvision

[7] P. Aguiar, L. Mendona, and V. Galhardo, "Opencontrol: A free opensource software for video tracking and automated control of behavioral mazes," *Journal of Neuroscience Methods*, vol. 166, pp. 66–72, June 2007.

[8] J.-E. Poirrier, L. Poirrier, P. Leprince, and P. Maquet, "Gemvid, an open source, modular, automated activity recording system for rats using digital video," *Journal of Circadian Rhythms*, vol. 4, June 2006.

[9] Music Technology Group - Pompeu Fabra University. (2008, June) reacTIVision 1.4. [Online]. Available: http://reactable.iua.upf.edu/?software

[10] M. D. Abramoff, P. J. Magelhaes, and S. J. Ram, "Image processing with imagej," *Biophotonics International*, p. 36.

[11] N. Correll, G. Sempo, Y. L. de Meneses, J. Halloy, J.-L. Deneubourg, and A. Martinoli, "Swistrack: A tracking tool for multi-unit robotic and biological systems," in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006)*. IEEE/RSJ, 2006, pp. 2185—2191.

[12] T. Lochmatter, P. Roduit, and N. Correll. (2008) SWISTrack (Wikibook). [Online]. Available: http://en.wikibooks.org/wiki/SwisTrack

[13] M. Fiala, "Artag, a fiducial marker system using digital techniques," *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, pp. 590–596 vol. 2, June 2005.

[14] C. M. Cianci, X. Raemy, J. Pugh, and A. Martinoli, "Communication in a Swarm of Miniature Robots: The e-Puck as an Educational Tool for Swarm Robotics," in *Simulation of Adaptive Behavior (SAB-2006), Swarm Robotics Workshop*, ser. Lecture Notes in Computer Science (LNCS), 2007, pp. 103–115.

[15] J. Halloy, J.-M. Amé, G. S. C. Detrain, G. Caprari, M. Asadpour, N. Correll, A. Martinoli, F. Mondada, R. Siegwart, and J.-L. Deneubourg, "Social integration of robots in groups of cockroaches to control self-organized choice," *Science*, vol. 318, no. 5853, pp. 1155–1158, 2007.

[16] R. Y. Tsai, "A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses," *IEEE Journal of Robotics and Automation*, vol. RA-3, no. 4, pp. 323–344, August 1987.

[17] T. Lochmatter, X. Raemy, L. Matthey, S. Indra, and A. Martinoli, "A comparison of casting and spiraling algorithms for odor source localization in laminar flow," in *Proc. of the 2008 IEEE Int. Conf. on Robotics and Automation*, may 2008, pp. 1138—1143.